System Design Document for android application DrinkIT

Kajsa Bjäräng, Viktoria Enderstein, Elin Eriksson, Lisa Fahlbeck,Alice Olsson
19/10-2018
version 3

1 Introduction

DrinkIT is a pre party game application designed for groups of two or more. The game revolves around communication between players. The game's purpose is to create fellowship for new such as old friends between. By doing the challenges DrinkIT gives you, you will be ready to take over the night with your now closer friends.

1.1 Design goals

The goal of the application is to make a functional game with few dependencies that is easy to reform and develop with loose coupling. The classes should all have one responsibility and not interfere with classes they have nothing to do with and should therefore be isolated and encapsulated.

1.2 Definitions, acronyms, and abbreviations

DrinkIT - The name of the application as well as the main class in the model of the application
Challenge -
MVC - Design pattern, Model View Controller.
Gradle -
GameRound -
Travis -
Android Studio - An Integrated development environment (IDE). It's the official IDE for Google's Android Operating system.

2 System architecture

The application is made in Android Studio, written in the language Java and uses the built in animation tool Gradle which comes included in Android Studio. DrinkIT is an interactive game for two to ten players, aming to create the perfect prepary experience. The application is played on a single unit.

2.1 Game flow

To start the game the user has to add the name of all participating players, then choose which categories should be included and the approximate duration of the game. Thereafter a challenge from one of the chosen categories is randomly shown. Most challenges are aimed to a specific player, who´s name will the show up together with the challenge. Other challenges are aimed for the whole group playing. Some challenges gives the player points if they are accomplished whilst others are just for fun. After the number of challenges corresponding to the chosen duration is played the game is finished with a scoreboard and an option to continue playing.
.

2.2 User stories

The main user stories used to achieve the overall flow of the application are the following:

2.2.1 As a user, I want to add a player, so I can have multiple players.

Acceptance Criteria:

- The player exists in the list of players
- It is not possible to add the same player twice

Tasks:

- There is a form to enter a name
- There is a button to add the player

2.2.2 As a user, I want to be able to choose categories, so I can customise the game with challenges I like.

Acceptance Criteria:

- It is possible to see all different categories
- It is possible to only choose one category
- It is possible to choose multiple categories
- It is possible to choose any number of categories
- Only challenges from the chosen categories will be shown during the game.
- It's possible to remove a chosen category

Tasks:

- There exists a number of categories
- If a category is chosen there is a visual feedback
- There are different buttons to add different categories to the game
- There is a button to chose all categories immediately

2.2.3 As a user I would like to choose an approximate duration of the game, so I can decide for how long I would like to play.

Acceptance Criteria:

- ❖ When a game is started an option to choose the duration of the game is displayed.
- ❖ There are three options to choose from
- ❖ Three of the options are set amounts of rounds (short, middle, long game)
- ❖ Depending of the number of players and the chosen option for duration an even number of rounds is calculated.
- ❖ The calculated number of rounds, depending of the number of players and the chosen option is the exact number of rounds played before the game is finished.

Tasks:

- ❖ There are 3 buttons to choose the duration of the game
- ❖ The number of rounds is set according to the chosen duration button and the amount of players.

2.2.4 As a user, I want to see a challenge when I play so I can do the challenge.

Acceptance Criteria:

- ❖ I can see a card on screen each round
- ❖ A player is chosen to play each specific challenge
- ❖ The difficulty of the challenge is shown on screen
- ❖ The challenge shown belongs to one of the chosen categories
- ❖ The challenge shown has not been shown before during the active game

Tasks:

- ❖ Randomize a player
- ❖ Show the name of the player on the display.
- ❖ Randomize a challenge.
- ❖ Show the task on the display

2.2.5 As a user, I want to see a clear, intuitive startpage with I open the app, so I can easily see how to continue.

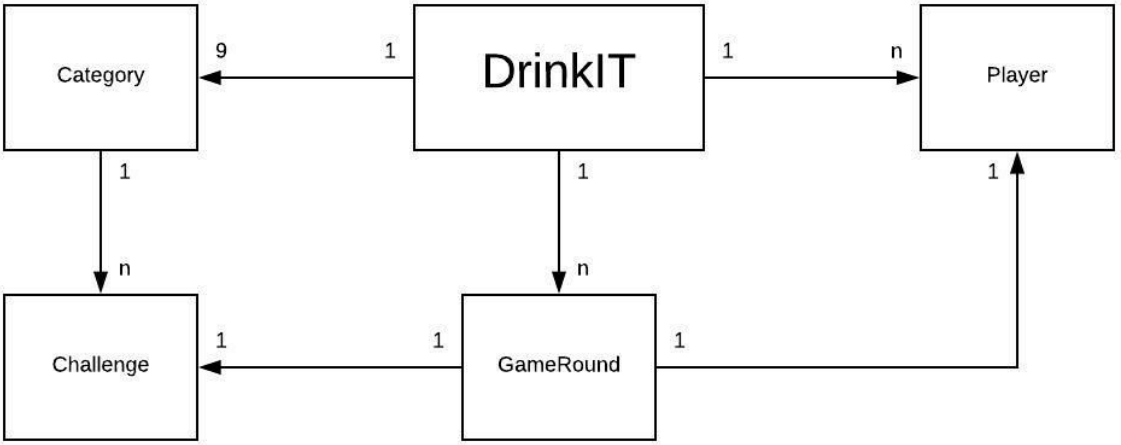
Acceptance Criteria:

- ❖ On the startpage there is a “start the game” button that when chosen lets the user enter all required information to start a new game
- ❖ On the startpage there is a “how to play” button that when chosen informs the user how the game works

Tasks:

- ❖ There is a button to continue to the next page
- ❖ There is a button to see the instructions of the game

2.3 Domain model



The application consists of a single component. The model package of the application is represented in the domain model. DrinkIT is the main class in the model and handles most of the funktionality for this package.

GameRound is the class with connects a challenge and a player to make a Gameround. The class lies in the model part of the application and sends the Player and the Challenge to the DrinkIT class.

DrinkIT contains a list of players which has a name and a point. These players is created when the player writes the names in the beginning of the game. DrinkIT also contains a list of nine categories. These categories are subclasses to the abstract class Category and are put to active if the player chooses to include the category in the game. These subclasses also contain a list of challenges. These challenges are one of three subtypes to the abstract superclass Challenge and are adapted differently to how the view should be.

The class GameRound gets a random player and a random challenge from one of the active categories. These are sent to DrinkIT which are connected to the controller which is connected to the view and the information reaches the user. DrinkIT saves a list of all played gamerounds to be able to keep statistics and keep track that not the same user get the same challenge twice.

3 System design

The application is designed in order to make it uncomplicated for future developers to understand, navigate and reform the code. To achieve this the system architecture strive for the principles Single Responsibility and Open Closed Principle the application is open for expansion and closed for modification.

3.1 Principles and patterns

Object oriented programming have different types of design patterns to solve common recurring problems. The patterns are used as a help to follow different principles for object oriented code.

3.1.1 Single responsibility principle

Single responsibility stands for do one thing and do it well. Which leads to a robust code. The consequence becomes that if changes need to be done in the code the chance other parts will get affected as well in the code is smaller. With this in the back of our minds we have tried to accomplish this throwout the whole code as good as possible.

3.1.2 Open closed principle

Open closed principle is a principle that stand for a code that is open for extension but closed for modification. Which means if the code is going to expand you don't need to change in several place in the code each time. This principle has been something we have tried to achieve during the whole code.

3.1.3 Encapsulation

Encapsulations goal is to avoid global access of variables and methods which prevents data to be modified by an outside user. Which leads to a more legible code and easier to expand which been implemented in the code.

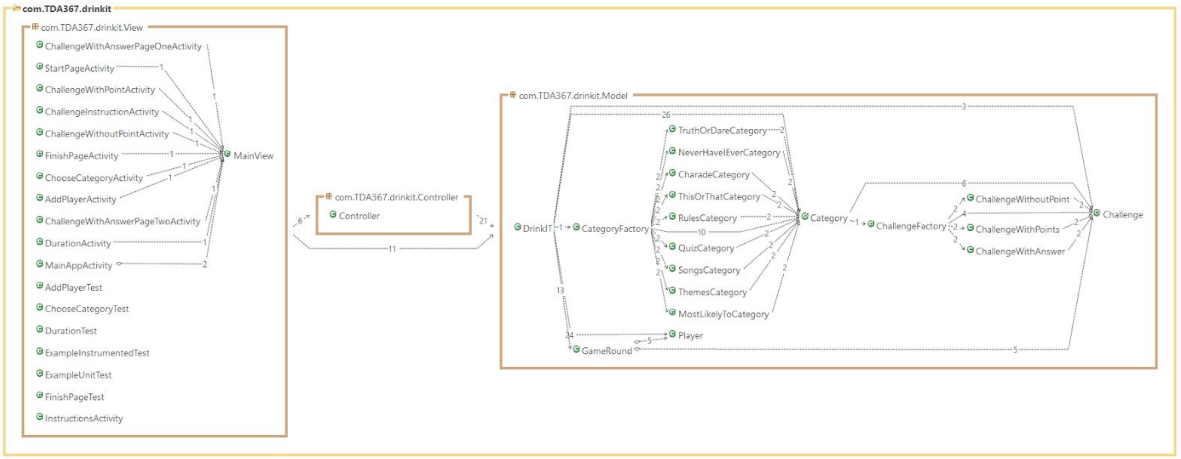
3.1.4 Factory pattern

The goal with factory patterns is to hide internal implementation. It's a factory to create similar objects and act like a more sophisticated constructor. With the factory you decrease the amount of dependencies and makes it easy to expand the code.

3.1.5 MVC

MVC is a pattern usually used when an application contains a type of view. The goal of the dependency is to achieve as high cohesion and low coupling as possible. Some guidelines is that the view should be dumb and don't depended on neither the controller or the model. The model should contain the domain logic and the controller should only contain external input. Preferably there should even be an application which should depended on each package and nothing should depend on the application.

3.1.6 Dependency analysis



3.2 Design model

• Describe your design model (which should be in one package and build on the domain model)
• Give a class diagram for the design model.

The design model is **uppbyggd** in three different packages, View, Model and Controller and follows, as earlier mentioned, the design pattern MVC. The class DrinkIT which lies in the Model package, holds a key position since DrinkIT is the gate between the Controller and the Modell. It is possible to say that the DrinkIT class is the heart of the game since it holds almost all of the underlying game logic.

The usage of two factories comes from the fact that the two classes Challenge and Category are made abstract and therefore impossible to instance. Since they both also are superclasses the implementation of the Factory pattern came naturally

Following the Factory pattern helps to hide intern implementation and reduces dependencies and code duplicering decrease.

work in progress:
Som tidigare nämnt är Factory pattern bra att använda för att dölja intern implementation och vi har därmed valt att implementera detta mönster för de två superklasserna Catergory och Challenges. Inga av dessa två ska gå att skapa och är därmed abstrakta och endast subtyperna går att skapa. Att använda factory pattern gör också att beroenden och kod duplicering minskar.

MainAppActivity initialiserar hela spelet och skapar en instans för varje package som sedan används i resten av spelet. MainView skapas i MainAppActivity för att minska beroendet i View-paketet genom att bara skapa controllern en gång istället för i varje activity.

3.3 Quality (test) //eller egen punkt 4

Efter varje utförd UserStory är tester implementerade. Dessa går att finna i mappen com.TDA316.drinkit.View.
• List of tests (or description where to find the test)
• Quality tool reports, like PMD (known issues listed here) Ramverk typ Junit printscreen vart man hittar testerna //Jacoco???? kollar upp deadcode

Diagrams
• Dependencies (STAN or similar)
• UML sequence diagrams for flow.

NOTE: Each Java, XML, etc. file should have a header comment: Author, responsibility,
used by ..., uses ..., etc.

4 Persistent data management
Textfiles is used for the categories. Each textfile contains the challenges for that specific category. One of the first things that happens in MainAppActivity is that the textfiles gets scanned by their pathname. They can be found in the map “res” under “raw”. The textfiles then gets splitted up into a list of challenges for each category.

5 References
Här är exempel SDD:
https://github.com/Tejpbit/TDA367/blob/master/Documents/RAD%26SDD/SDD.pdf
https://github.com/nahojjjen/OOP-Project-TowerDefence/blob/master/document/SDD.pdf